Lessons Learned in the Sierra Center of Excellence Migrating to Heterogenous Computing

David Richards, Ian Karlin, Rob Neely

Sept 2, 2020

LLNL-PRES-813866

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Acknowledgements - This talk builds on the work of many

Thank you to co-authors, code teams, support staff, COE Vendors, and everyone else who has helped to make Sierra a success

Johann Dahm
Aaron Black
Adam Bertsch
Leoplod Grinberg
Ian Karlin
Sara Kokkila-Schumacher
Edgar Leon
Rob Neely
Ramesh Pankajakshan
Olga Pearce
Brian Ryujin

Jason Burmark
Brian Pudliner
Adam Kunen
David Dawson
Rich Hornung
David Beckingsale
Peter Robinson
Tom Scogland
Holger Jones
David Poliakoff
Jim Glosli

Bert Still
Katie Lewis
Bruce Hendrickson
Matt Cordery
David Appelhans
Steve Rennich
Max Katz

And Many Others…

Summit and Sierra Supercomputers

The Sierra Center of Excellence was a close partnership between the NNSA, IBM, and Nvidia

- Established joint work plans, information sharing, and collaboration mechanisms
- Dedicated vendor staff worked alongside lab code teams
  - Some staff assigned to work at lab sites
- Labs provided access to our codes
  - Including classified codes for those with security clearance
- Vendors provided NDA information and early access to hardware and software

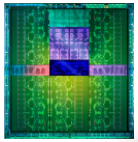
Forming a Center of Excellence has become a recognized best practice for large DOE system procurements

Sierra is LLNL's first heterogeneous HPC system

Components

IBM POWER9
- Gen2 NVLink

NVIDIA Volta
- 7 TFlop/s
- HBM2
- Gen2 NVLink



Mellanox Interconnect
- Single Plane EDR InfiniBand
- 2 to 1 Tapered Fat Tree

Compute Node

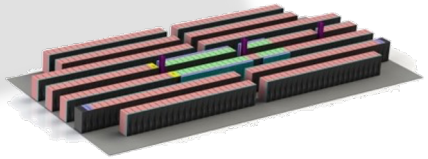
2 IBM POWER9 CPUs
4 NVIDIA Volta GPUs
NVMe-compatible PCIe 1.6 TB SSD
256 GiB DDR4
16 GiB Globally addressable HBM2 associated with each GPU
Coherent Shared Memory

Compute Rack

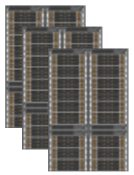
Standard 19"
Warm water cooling

Compute System

4320 nodes
1.29 PB Memory
240 Compute Racks
125 PFLOPS
~11 MW



GPFS File System

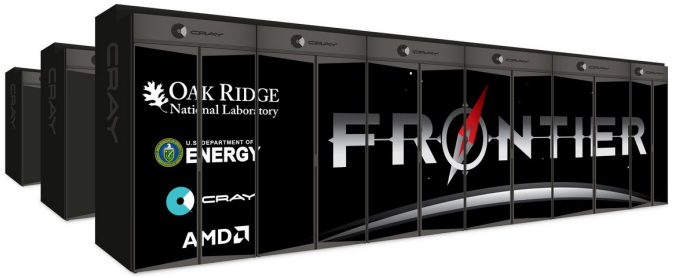


154 PB usable storage
1.54 TB/s R/W bandwidth

Recently announced DOE systems clearly show we have now entered the heterogeneous era

Perlmutter NERSC, 2020
AMD CPU, Nvidia Tesla GPU

Frontier ORNL, 2021
AMD CPU, AMD GPU, 1.5 ExaFlop

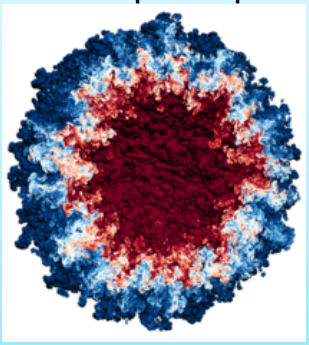


Aurora Argonne, 2021
Intel CPU, Intel Xe GPU, > 1 ExaFlop

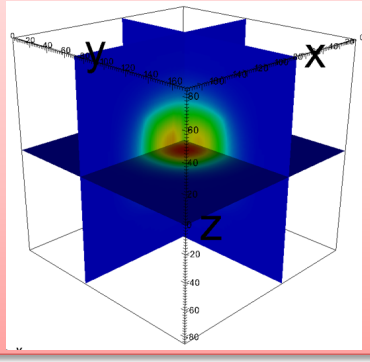
El Capitan LLNL, 2022
AMD CPU, AMD GPU, > 1.5 ExaFlop

Our switch to GPU-based computing is paying off with big performance increases

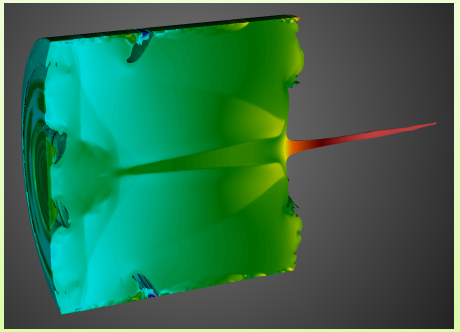
Ares, RT Mixing
13x speedup



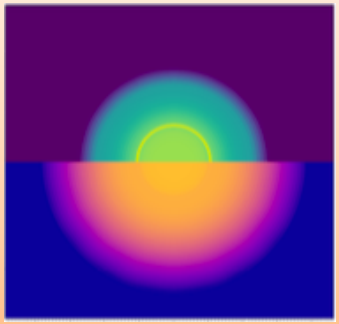
Ardra, Reactor Safety
16x speedup



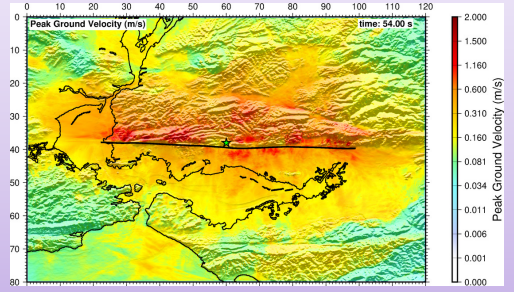
ALE3D, Shaped Charge
8x speedup



Kull/Teton
Radiating Sphere
7x speedup



SW4, Hayward Fault, 28x speedup

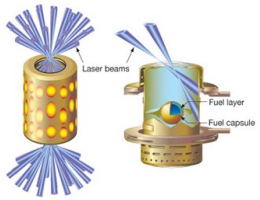


Large, integrated multi-physics codes provide simulation capabilities for a broad range of application domains

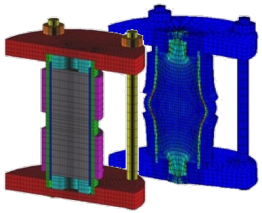
- Millions of lines of code in multiple programming languages
- Scale to O(1M) MPI ranks
- Multiple spatial/temporal scales
- Maintain connection to prior V&V efforts
- Coordinate with 10-60+ libraries

- Long life-time projects
  - 15+ years of development by large teams
  - 10–20+ people, ~50/50 CS/Physicists
- Portable performance
  - Our codes must be fast, reliable, and accurate on multiple systems
  - Laptops, Workstations, Commodity Clusters, Advanced Architectures, Heterogenous Architectures

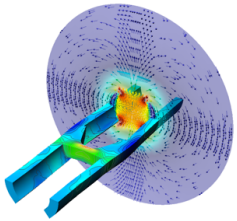
Inertial Confinement Fusion



HE Cookoff

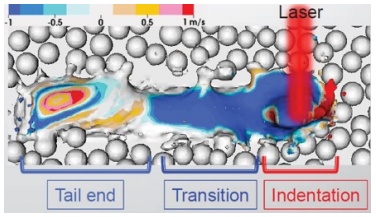


Navy Railguns



Fracture and Failure

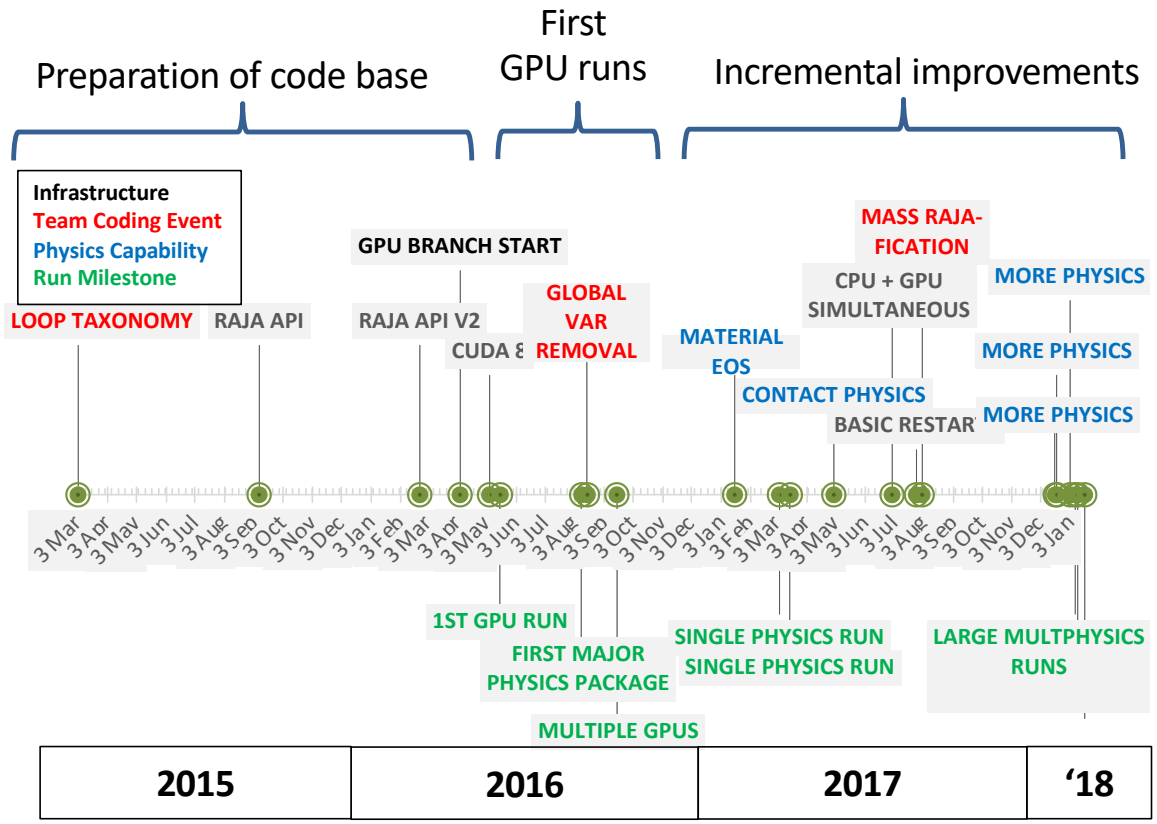
Additive Manufacturing



Our codes are specifically tailored to our mission space and HPC capabilities presenting unique challenges

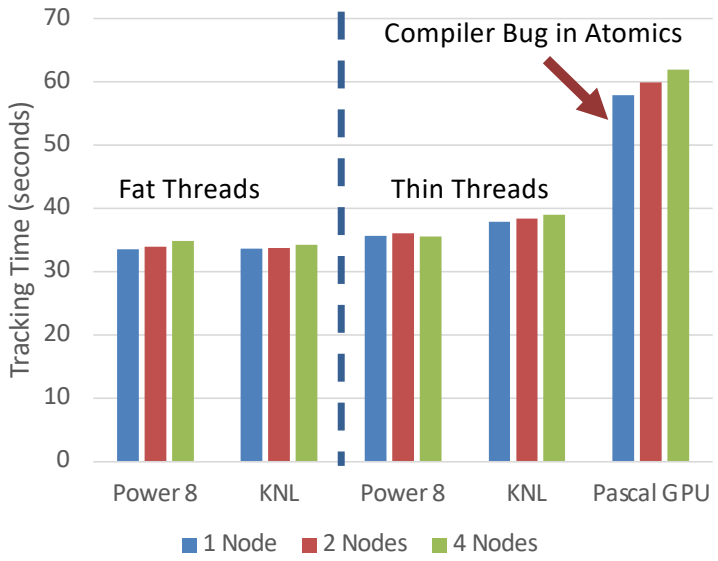
Successful application modernization follows a consistent pattern

1. Refactor and remove anti-patterns
2. Create a mini-app to explore design space
3. Use portable abstractions and frameworks
4. Focus on a specific use case
5. Search for additional parallelism
6. Manually manage memory
7. Iteratively apply the steps above



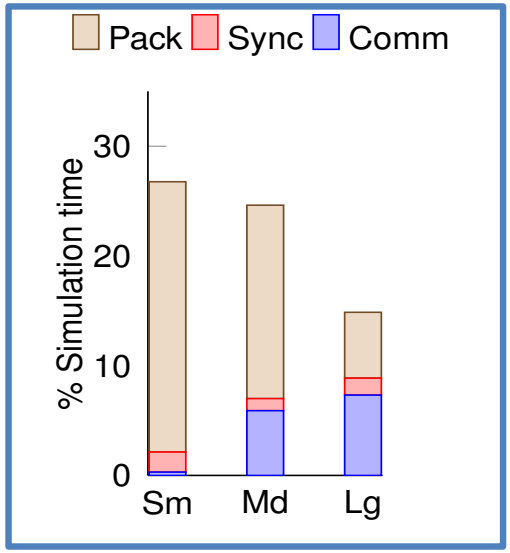
Proxy apps are extremely useful to explore design and refactoring choices as well as performance bottlenecks

Quicksilver tracking loop times
(weak scaling, lower is better)



Testing thread strategies for Mercury

Comb represents the data packing and communication for the halo exchange in Ares



Messaging time breakdown for various job sizes on Sierra.

Proxy apps are also useful for benchmarking and vendor co-design

RAJA is our performance-portability solution that uses standard C++ idioms to target multiple back-end programming models

- Decouple loop traversal and iterate (body)
- An iterate is a “task” (aggregate, (re)order, …)
- IndexSet and execution policy abstractions simplify exploration of implementation/tuning options without disrupting source code

Pattern
(forall, reduction, scan, etc.)

Execution Policy
(how loop runs: PM backend, etc.)

Index
(index sets, segments to partition, order, …. iterations)

C-style for-loop

```
double* x ; double* y ;
double a, tsum = 0, tmin = MYMAX;

for ( int i = begin; i < end; ++i )  {
     y[i] += a * x[i] ;
     tsum += y[i] ;
     if ( y[i] < tmin ) tmin = y[i];
}
```

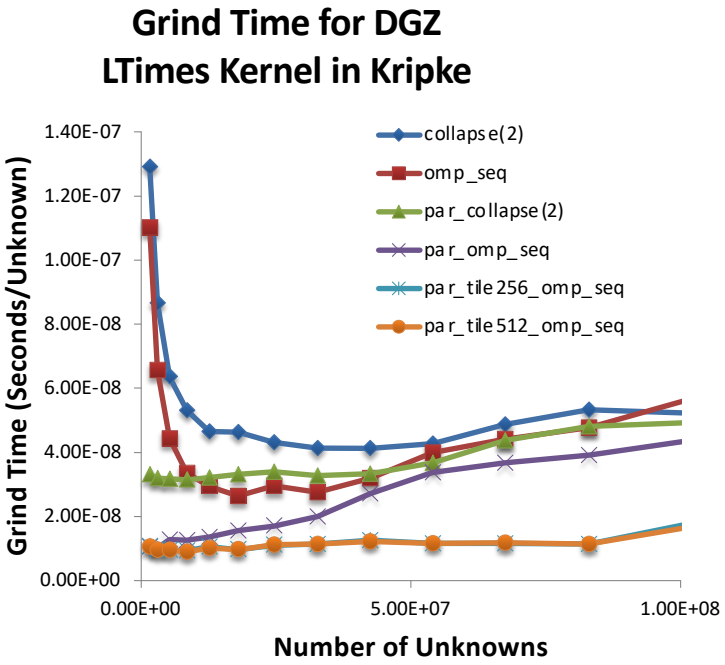
RAJA-style loop

```
double* x ; double* y ;
double a ;
RAJA::SumReduction<reduce_policy, double> tsum(0);
RAJA::MinReduction<reduce_policy, double> tmin(MYMAX);

RAJA::forall< exec_policy > ( index_set , [=] (int i) {
     y[i] += a * x[i] ;
     tsum += y[i];
     tmin.min( y[i] );
}  );
```

RAJA allows us to write-once, target multiple back-ends

Kripke was an essential tool to explore design patterns for Arda and co-design RAJA



Exploring execution policies for Ardra

RAJA style nested for-loop

```
RAJA::View vview(v_ptr,
                 make_perm_layout(ni,nj));
//...

RAJA::forallN< exec_policy, INDX, JNDX >(
               RangeSegment(1, ni),
               RangeSegment(0, nj),
               [=](INDX i, JNDX j) {

   vview(0, j) += vview(i, j);

});
```

The RAJA nested loop abstraction generates optimal loop ordering for any runtime parameters

Unified (coherent) memory is helpful, but is not a panacea

No single strategy: multiple paths to success have emerged

- SW4: Allow managed memory to handle transfers. Overhead amortized by much re-use between transfers.
- Ares: Data transfers are explicit for performance. Managed memory pointers are helpful for libraries and code simplicity.
- Teton: All data transfers are explicit.

Abstractions improve code performance and developer productivity

- CHAI: Smart pointers automate explicit data transfers (Ardra, ALE3D)
- Umpire:
  - Unified, portable API to 3rd party memory capabilities
  - Coordinates memory use/introspection among multiple packages
  - Provides memory pools etc. to improve performance

Host-device data transfers must be treated as first class concerns

Umpire is being developed to coordinate complex memory allocations and movement

Assume three packages/libraries A,B,C – each with their own view of the GPU memory resources

Phase 0

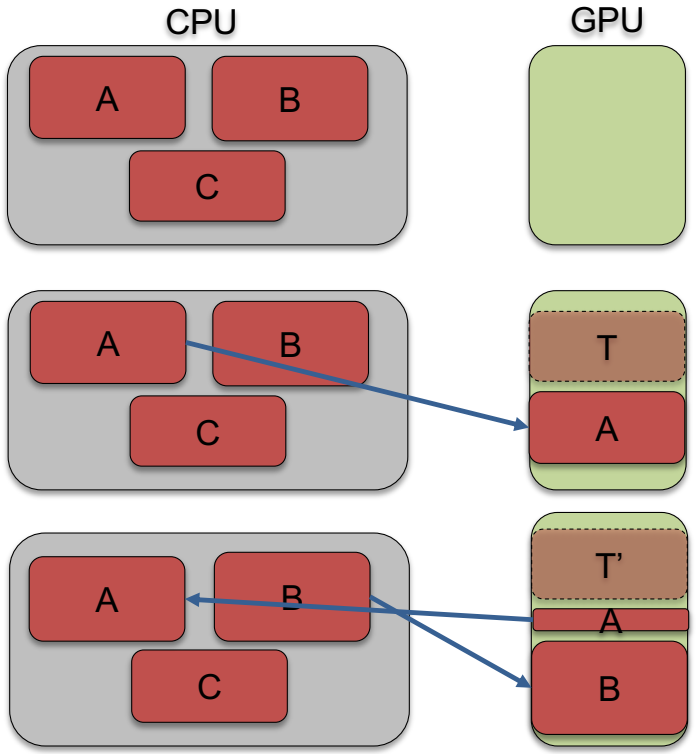
- Initial state of problem staged in CPU memory

Phase 1

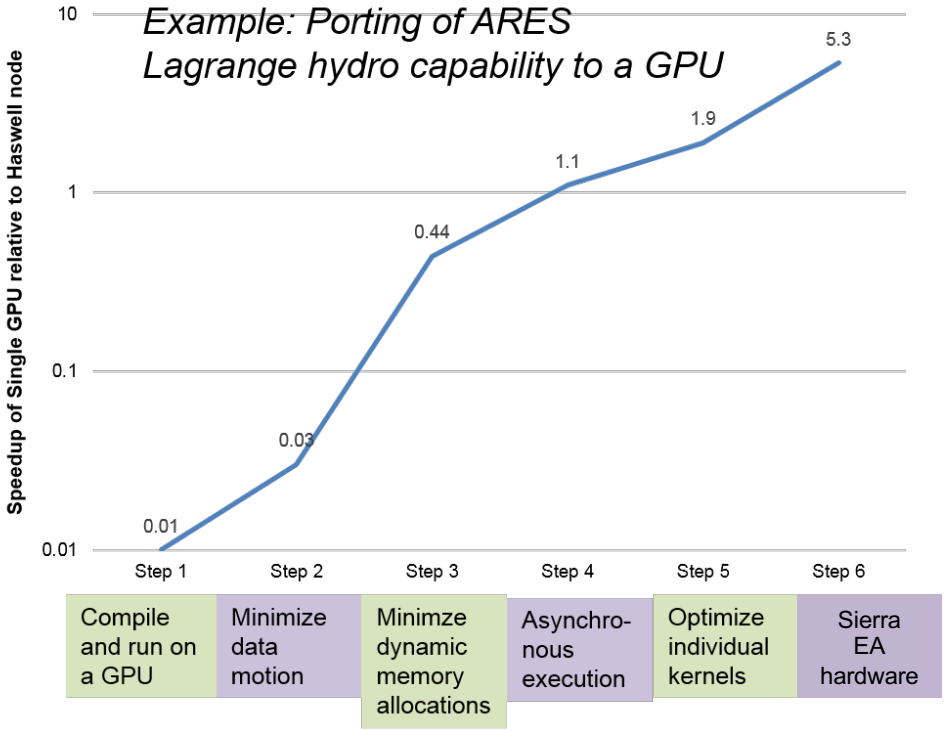
- A executes first
- A allocates temporary data in a memory pool (T)
- A's data is copied to GPU

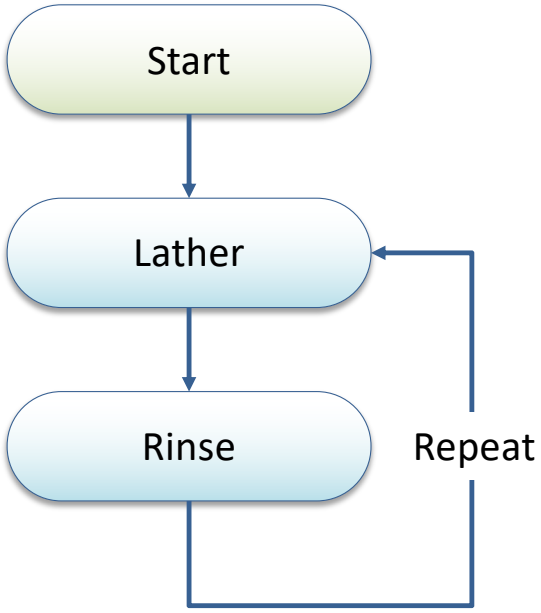
Phase 2

- A's data copied back to CPU
  - Some shared data remains on the GPU
- B's data copied to GPU
- Temporary data T deallocated
- B allocates temporary data T' using the same memory pool



Performance improvement is an iterative process. Each step improves performance but also uncovers the next problem





This result required sustained effort over long time by many people.
Vendor partners and COE were critical to this success.

Fortran/OpenMP is not as well supported as C/C++ on GPUs

- Flang/F18 is likely to help with compiler availability
- OpenMP is the only real choice for portable GPU off-load in Fortran. No mechanism for abstraction layers.
- Modern Fortran and features not shared with C/C++ such as shaped arrays or array notation are especially problematic
- Write your Fortran code as much like C as possible if you want it to perform well
- Up-to-date proxies and tests are critical to ensuring compilers will function as desired

The Fortran community is relatively small compared to C++. We should pool effort and spread the overhead/effort.

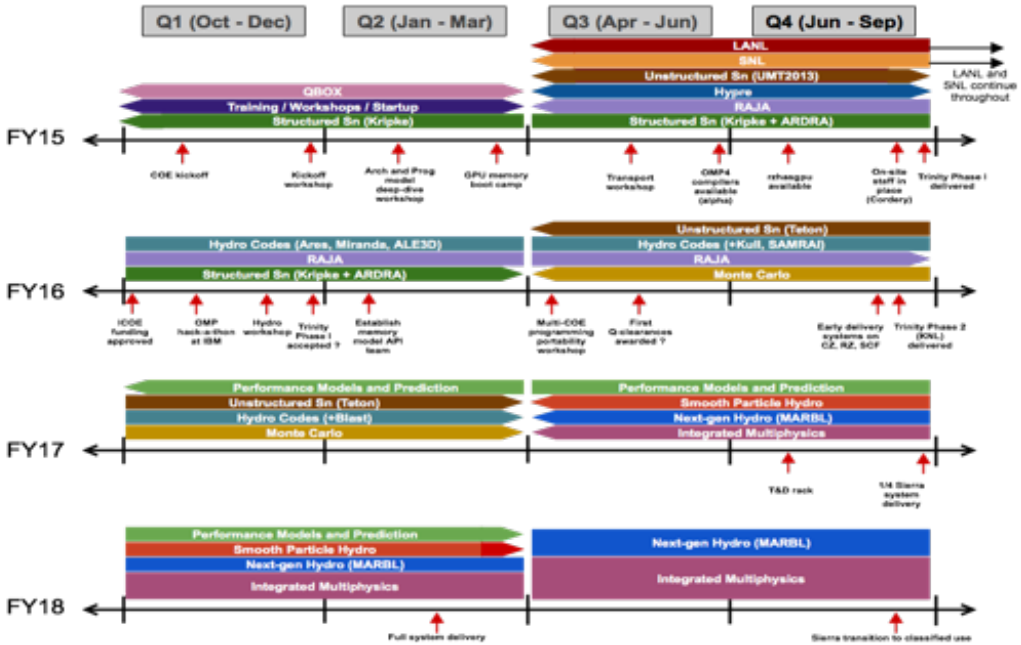
Complex workflows including machine learning, and real-time analytics or visualization are placing new demands on Sierra

Four key lessons learned from large scale workflows

- Optimize resource allocations at the workflow level
  - Consider which workflow elements benefit most from available hardware
  - Allocate data generators close to corresponding data consumers
- Use workflow management tools
  - Matching the available resources to ready tasks requires dedicated management software
  - Checkpointing a workflow can be harder than you think
- Consider the memory hierarchy and data sharing tools when designing a workflow
  - File I/O is not adequate to coordinate complex workflows
- Package managers and continuous integration can help ensure the reproducibility of a workflow

Sierra Center of Excellence work plans featured built-in flexibility

- Codes and libraries “phased in” over time
  - Multi-year plan overlaid with hardware and compiler availability to guide work plans
- Work plans were intentionally overcommitted to allow agility
- Earliest work focused on training and proxy apps
- After year one – pivoted to real applications and greater team engagement with vendor help



Effective collaboration doesn't happen by accident

- Schedule activities to ensure two-way engagement
  - Trying to force interactions when priorities don't align is a recipe for failure
- Be prepared to deviate from your plan
  - Things will go wrong
  - Opportunities will arise
- Invest in collaboration and software engineering tools
  - A common set of repos and communication tools will enhance productivity
  - Multi-site, secure tools can be hard to find
  - Avoid fragmentation of information
- Build multidisciplinary teams
  - Co-locate teams as much as possible.

It’s not all sunshine, lollipops, and rainbows

- Usual new system pains: MPI, scheduler, compilers. Most are largely resolved
- Tension between system stability and bleeding-edge system software
- Some apps/algorithms aren’t there yet
  - Monte Carlo, Multi-grid setup phase
- Proprietary tool suites don’t handle all of our use cases
  - Vendor tools don’t always play nicely with HPC, scaling, MPI, etc.
  - Open source tools provide choices for debuggers, performance tools, etc.
- Interoperability of parallel models (OMP, CUDA, …), compliers, memory handling, across libraries can be difficult

RADIUSS supports an LLNL-developed open source software stack for rapid and enduring HPC application development

Common Development Policies and Tooling	Training and Documentation	Leverage Programmatic Investments	Integrate into LLNL Applications
"Harden" for broad adoption	Encourage external adoption and contributions	Programs provide base long-term support and development	Provide path to both advanced architectures and software agility
Ease developer movement between projects	Not often a funded activity under research or program-specific projects	External outreach and community development	Long-term support and minor feature development

Product Categories

Build Tools → Physics and Math Libraries → Application CS Infrastructure → Data Management and Visualization → Portable Execution & Memory Mgmt → Performance Tools and Workflows

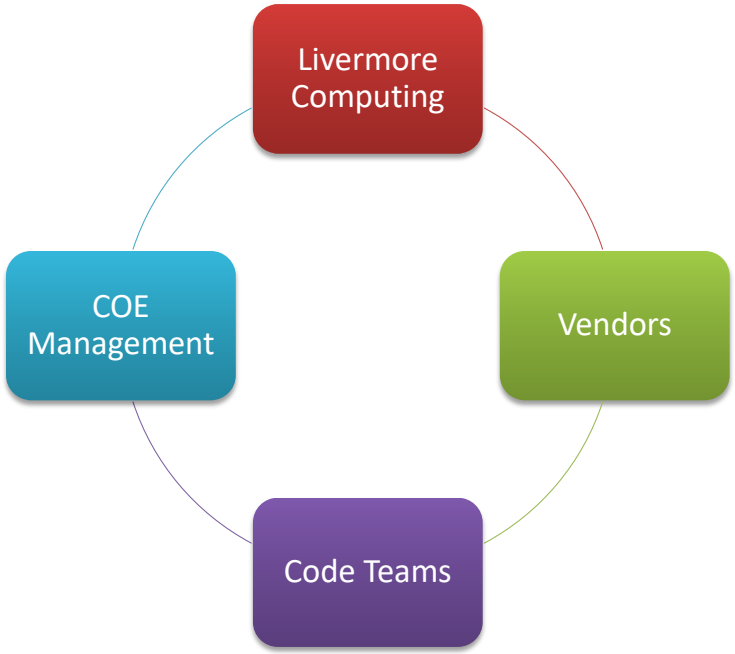
Foundational Concepts

Continuous Integration, Release Management, Integrated Testing, Deployment, Outreach

Software is core infrastructure to the laboratory, similar to institutional HPC platforms and laboratory space.
Sustained software investments are core to the mission of LLNL and our continued HPC leadership.

Porting to Sierra has taken years of hard work, but the results are worth it

- Many codes are seeing speedups of 10x or more
  - It *is* possible to incrementally refactor a large production code
- Code refactoring has reduced technical debt
  - But this takes commitment
- Increased performance is opening doors to previously impossible science
- Lessons learned and ecosystem improvements blaze a trail for others to follow
  - Future efforts should be easier/faster due to improvements in supporting software



Multi-discipline, multi-talent teams were essential to success on Sierra